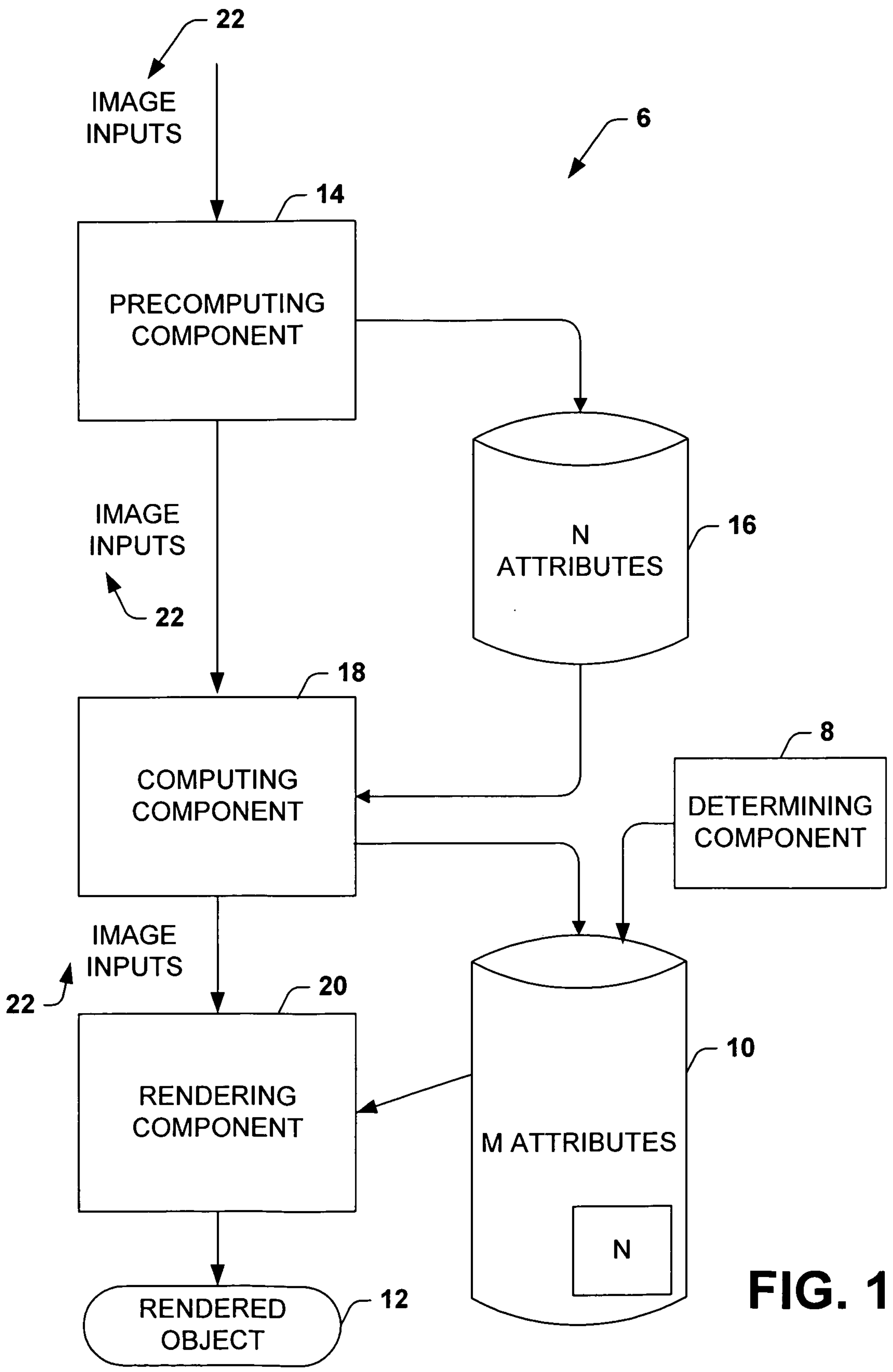
22
IMAGE INPUTS
6
14
PRECOMPUTING COMPONENT
N ATTRIBUTES
16
IMAGE INPUTS
22
18
COMPUTING COMPONENT
8
DETERMINING COMPONENT
IMAGE INPUTS
22
20
10
RENDERING COMPONENT
M ATTRIBUTES
N
RENDERED OBJECT
12

FIG. 1

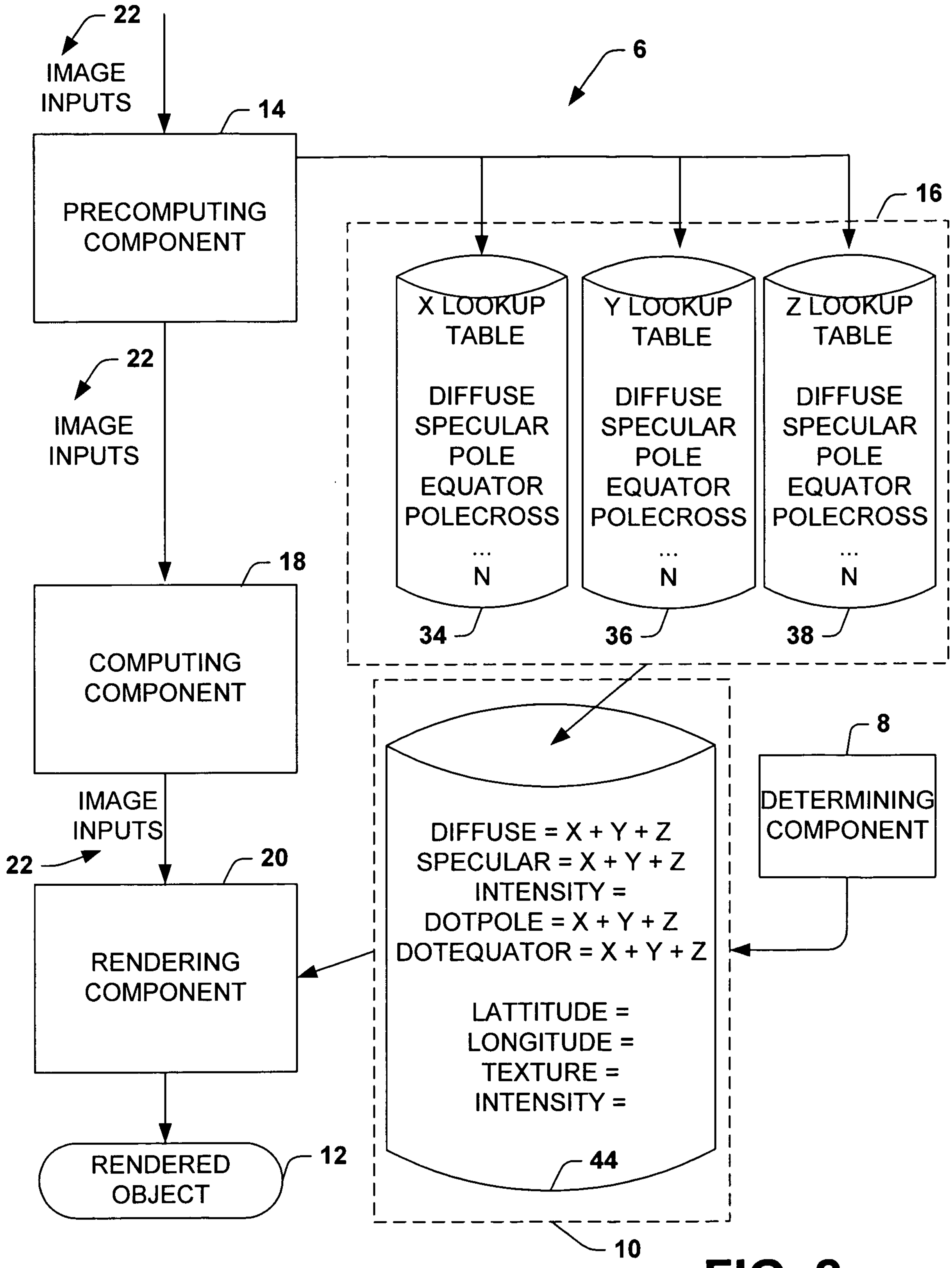
22
IMAGE INPUTS
6
14
PRECOMPUTING COMPONENT
16
X LOOKUP TABLE
DIFFUSE
SPECULAR
POLE
EQUATOR
POLECROSS
...
N
Y LOOKUP TABLE
DIFFUSE
SPECULAR
POLE
EQUATOR
POLECROSS
...
N
Z LOOKUP TABLE
DIFFUSE
SPECULAR
POLE
EQUATOR
POLECROSS
...
N
34
36
38
22
IMAGE INPUTS
18
COMPUTING COMPONENT
IMAGE INPUTS
22
20
RENDERING COMPONENT
DIFFUSE = X + Y + Z
SPECULAR = X + Y + Z
INTENSITY =
DOTPOLE = X + Y + Z
DOTEQUATOR = X + Y + Z
LATTITUDE =
LONGITUDE =
TEXTURE =
INTENSITY =
44
8
DETERMINING COMPONENT
RENDERED OBJECT
12
10

FIG. 2

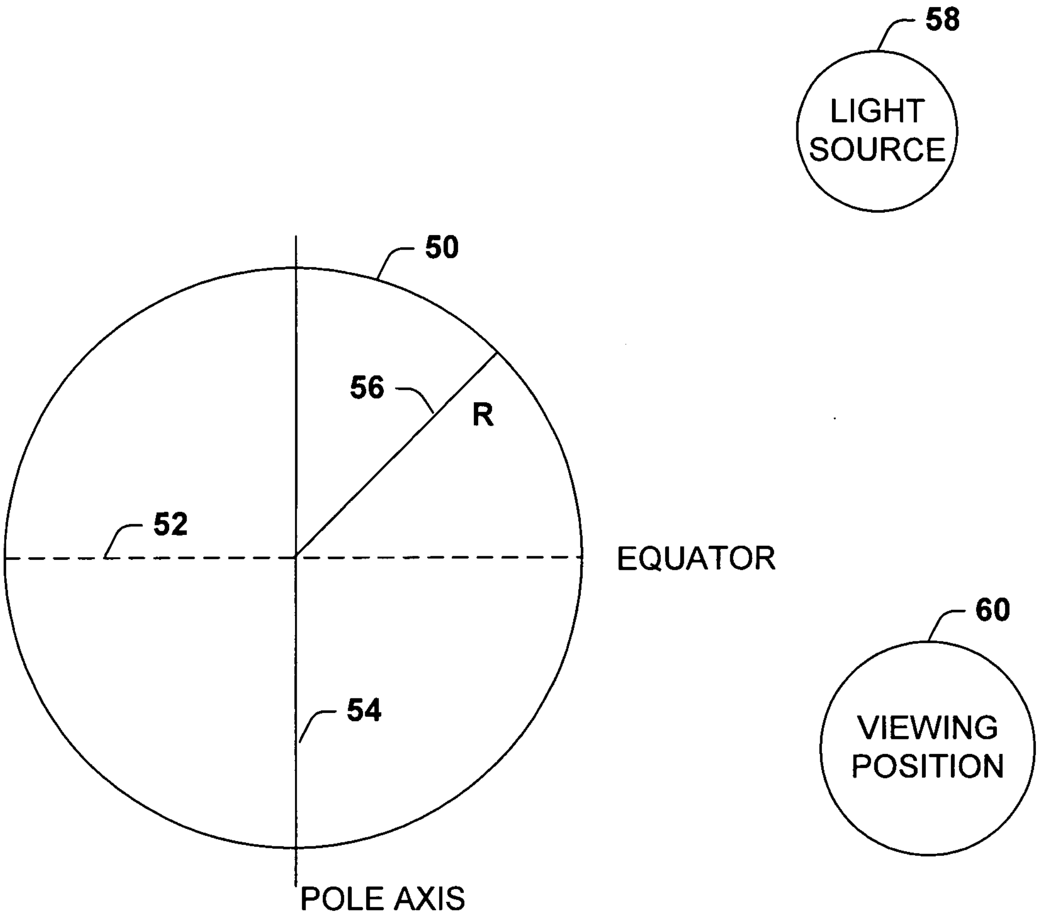
58
LIGHT SOURCE
50
56
R
52
EQUATOR
60
VIEWING POSITION
54
POLE AXIS

FIG. 3

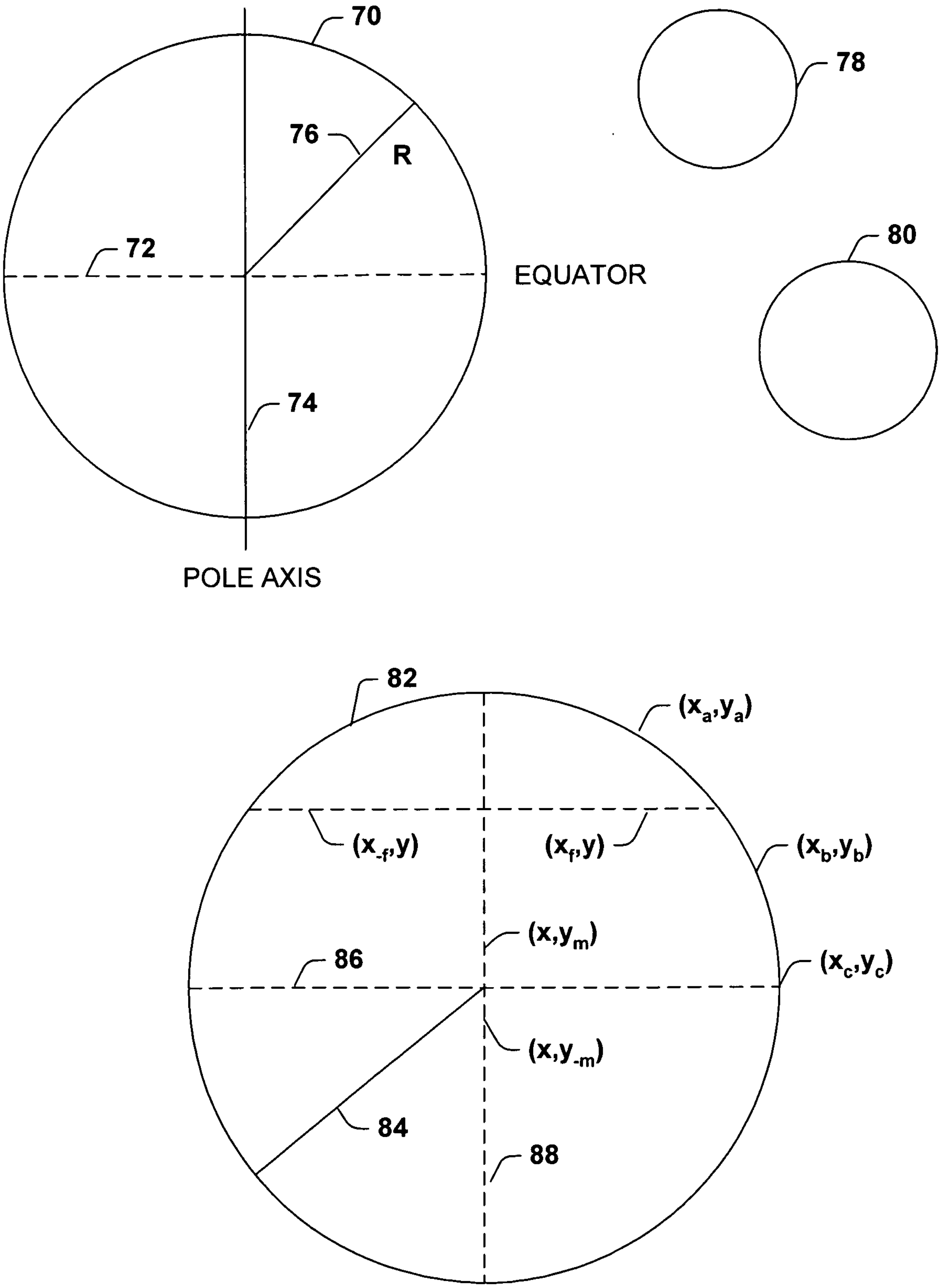
70
78
76
R
72
EQUATOR
80
74
POLE AXIS
82
$(x_a,y_a)$
$(x_{-f},y)$
$(x_f,y)$
$(x_b,y_b)$
$(x,y_m)$
86
$(x_c,y_c)$
$(x,y_{-m})$
84
88

FIG. 4

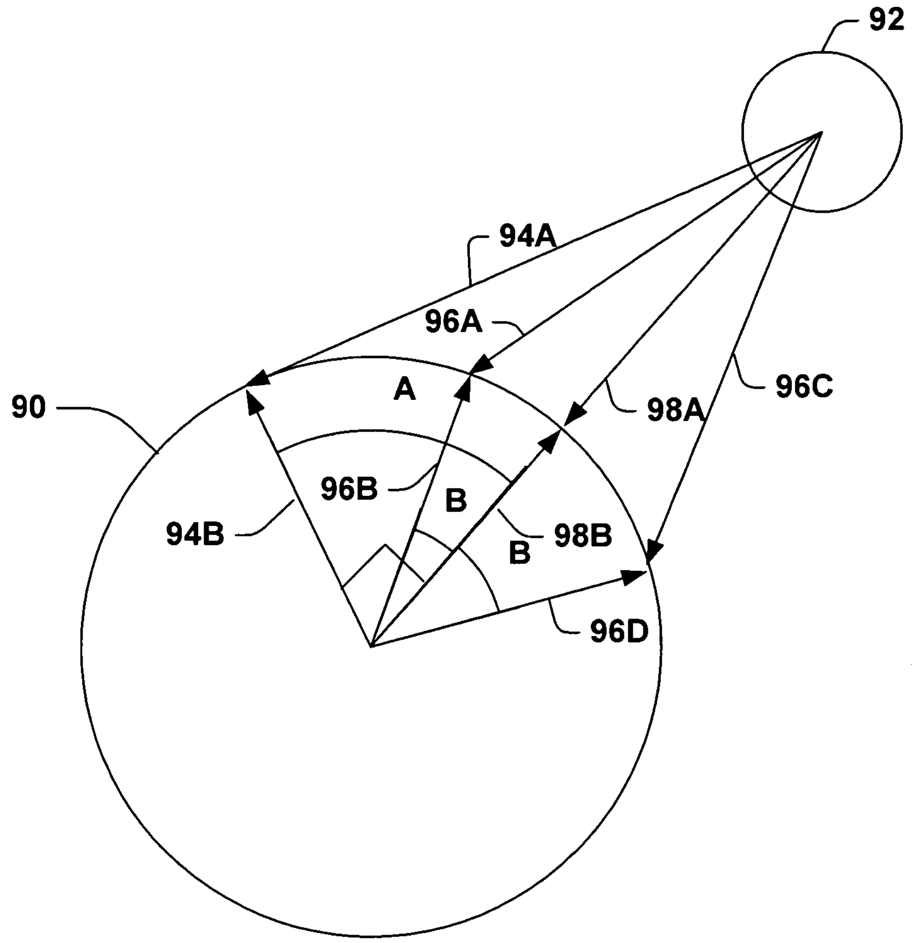

FIG. 5

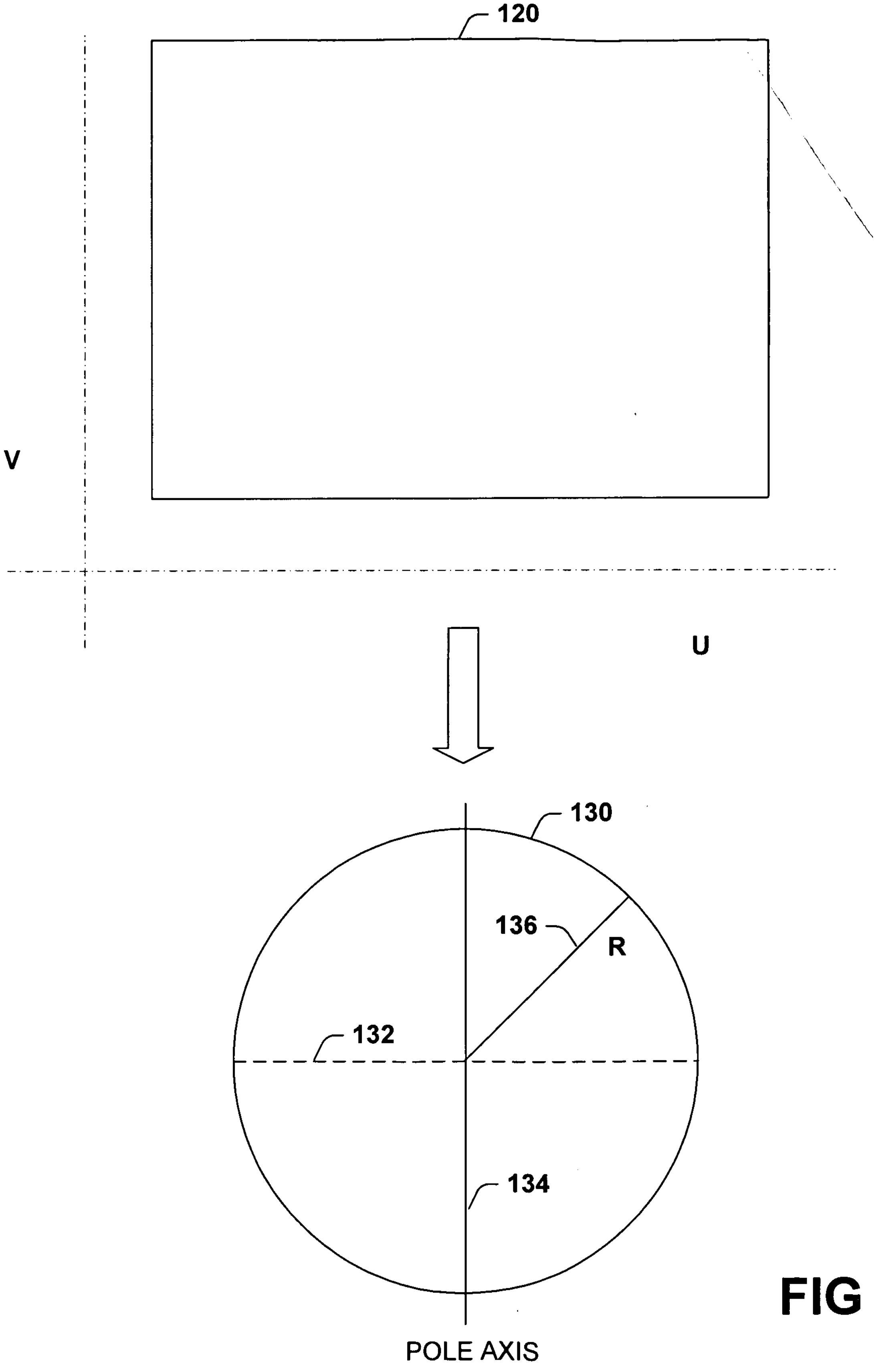

FIG 6

```
Draw A Sphere (input Radius, CenterX, CenterY, VectLight, VectViewer,
                              VectorPole, VectorEquator)
        // the image inputs include the size of the sphere, where it is to be drawn,
        // where a lighting source is positioned and where a viewer is positioned
{
        // set up initial vectors
        vectSpecularHighlight = Normalize(vectViewer + vectLight);
        vectPoleCrossEquator = VectorPole cross VectorEquator;

        // prepare lookup tables, can be computed before rendering
        // portions of later calculations pre-calculated here b/c x & y invariant to
        // other parameters
        for ( i = -rad; i <= rad; i++)
        {       j = i * 1 / rad;
                xMultiplyDiffuse[i] = j * vectLight.x;; // setup diffuse component
                yMultiplyDiffuse[i] = j * vectLight.y;
                xMultiplySpecular[i] = j * vectSpecularHighlight.x; // setup specular
                yMultiplySpecular[i] = j * vectSpecularHighlight.y;
                xMultiplyPole_LUT[i] = j * vectorPole.x;  // used for texture
                yMultiplyPole_LUT[i] = j * vectorPole.y;
                xMultiplyEquator_LUT[i] = j * vectorEquator.x; // setup equator
                yMultiplyEquator_LUT[i] = j* vectorequator.y;
                xMultiplyPE_LUT[i] = j * vectPoleCrossEquator.x; // where pole &
                xMultiplyPE_LUT[i] = j * vectPoleCrossEquator.y;  //equator cross
        }
        for ( x = 0; x < rad; x++ )    // finite set of discriminants
        {       disc = r^2 - x^2;
                for ( y = 0; y < x; y++ )       // thus finite set of z values
                {       disc = disc - y^2;
                        if ( disc > 0 )
                        { zInvRad = 1 / (squareroot(disc) * radius;
                          zMultiplyDiffuse_LUT[disc] = zInvRad * vectLight.z;
                          zMultiplySpecular_LUT[disc] = zInvRad *
                                vectSpecularHighlight.z;
                          zMultiplyPole_LUT[disc] = zInvRad * vectorPole.z;
                          zMultiplyEquator_LUT[disc] = zInvRad * vectorEquator.z;
                          zMultiplyPE_LUT[disc] = zInvRad *
                                vectPoleCrossEquator.z;
                        } // end if
                } // end for y
        } // end for x
// proc cont'd on Fig. 7b
```

FIG 7A

```
// Iterate over the scanlines in the sphere
// combining the precomputed lookup elements as you go
// for each scan line
for ( y = -rad; y <= rad; y++ )
{       RadiusSubYSquare = r^2 - y^2;
        Bound = edgeBuffer[abs(y)]; // bound is the horizontal displacement from
                                                 // y axis
        for ( x = (-bound + 1); x <= bound; x++ )
        {
152             // iterate over every pixel in the scanline y
                disc = RadiusSubYSquare - x^2; // compute disc for look up table
                                                         // index
                diffuse = yMultiplyDiffuse[y] + xMultiplyDiffuse[x] +
                        zMultiplyDiffuse_LUT[disc];
                specular = yMultiplySpecular[y] + xMultiplySpecular[x] +
                        zMultiplyDiffuse_LUT[disc];
                specular = SpecularRemapLUT[specular]; // remap to range 0 -1.0

150             // compute the final intensity for a pixel
                intensity = diffuse * diffuseFactor + specular * specularFactor +
                                ambient * ambientFactor;
                // compute the u & v texture components for a pixel
                NormalDotPole = xMultiplyPole_LUT[x] + yMultiplyPole_LUT[y] +
                                        zMultiplyPole_LUT[z];
                NormalDotEquator = xMultiplyEquator_LUT[x] +
                        yMultiplyEquator_LUT[y] + zMultiplyEquator_LUT[z];
                latitude = arccos(NormalDotPole);
                vTexture = latitutde/PI;
                longitude' = NormalDotEquator / sine(latitutde);
                clamp longitude' to range -1.0 to 1.0
                longitude = arccos(longitude');

                // determine how longitude wraps around sphere
                if (xMultiplyPE_LUT[x] + yMultiplyPE_LUT[y] +
                        zMultiplyPE_LUT[disc] < 0 )
                {       uTexture = longitude;      }
                else
                { uTexture = 1 - longitude;        }

                // fetch a textured pixel from coordiante uTexture, vTexture
                // scale intensity of textured pixel by Intensity
                // draw the lit, texture pixel at location ( x + centerX, y + centerY)
        } // end for x
} // end for y
} // end proc
```

FIG. 7B

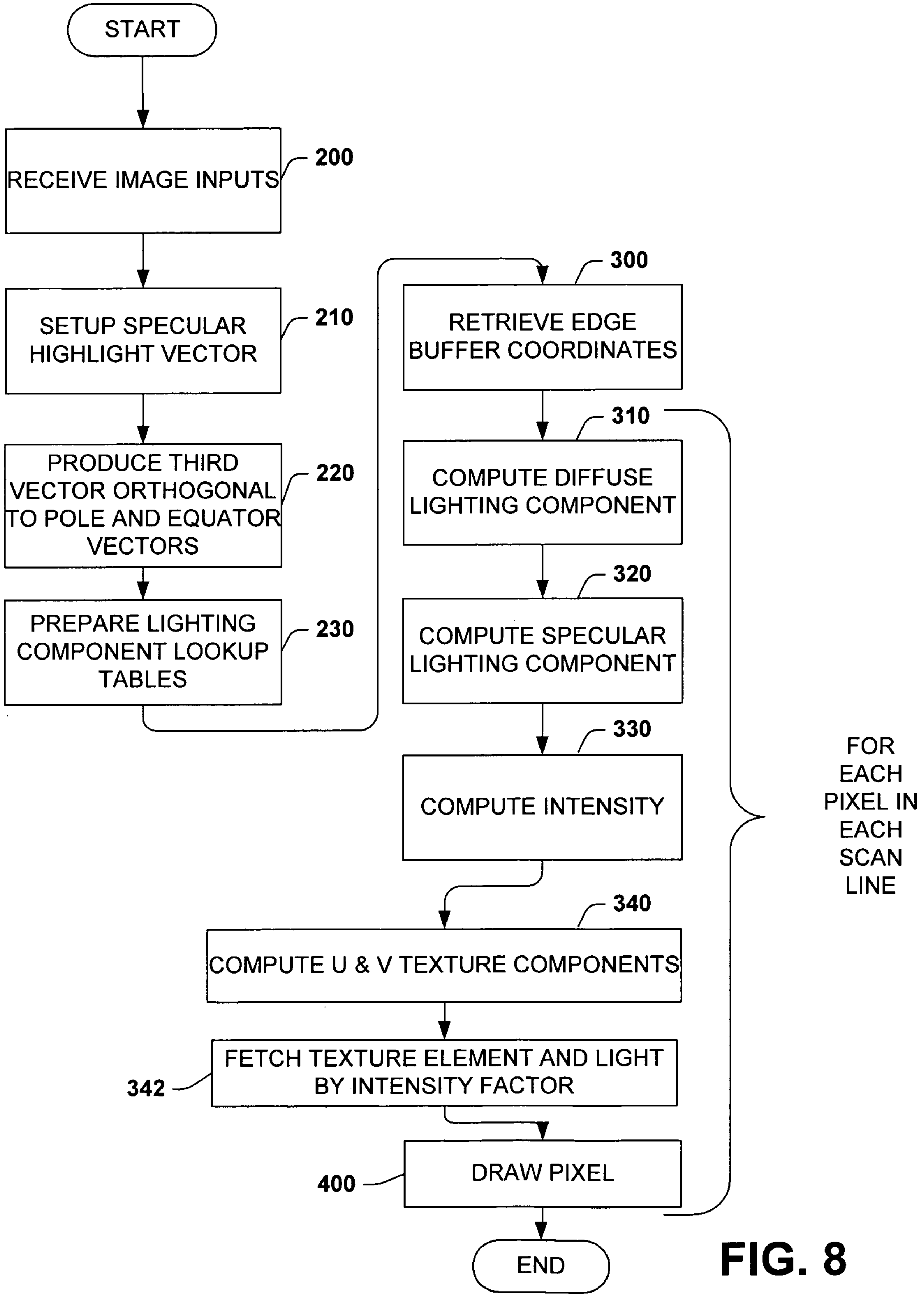
START
RECEIVE IMAGE INPUTS
200
SETUP SPECULAR HIGHLIGHT VECTOR
210
PRODUCE THIRD VECTOR ORTHOGONAL TO POLE AND EQUATOR VECTORS
220
PREPARE LIGHTING COMPONENT LOOKUP TABLES
230
300
RETRIEVE EDGE BUFFER COORDINATES
310
COMPUTE DIFFUSE LIGHTING COMPONENT
320
COMPUTE SPECULAR LIGHTING COMPONENT
330
COMPUTE INTENSITY
FOR EACH PIXEL IN EACH SCAN LINE
340
COMPUTE U & V TEXTURE COMPONENTS
342
FETCH TEXTURE ELEMENT AND LIGHT BY INTENSITY FACTOR
400
DRAW PIXEL
END

FIG. 8

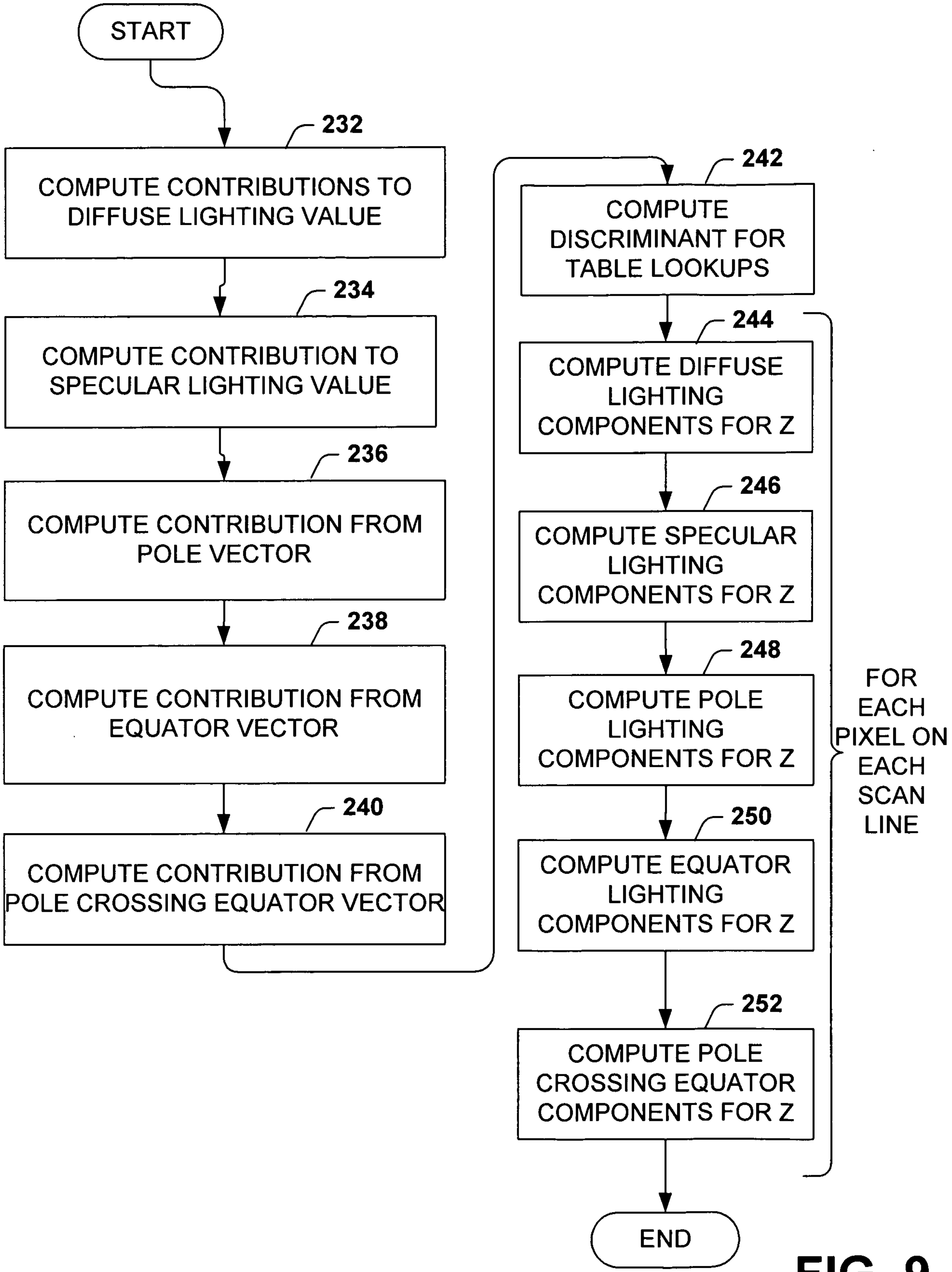
START
232
COMPUTE CONTRIBUTIONS TO DIFFUSE LIGHTING VALUE
234
COMPUTE CONTRIBUTION TO SPECULAR LIGHTING VALUE
236
COMPUTE CONTRIBUTION FROM POLE VECTOR
238
COMPUTE CONTRIBUTION FROM EQUATOR VECTOR
240
COMPUTE CONTRIBUTION FROM POLE CROSSING EQUATOR VECTOR
242
COMPUTE DISCRIMINANT FOR TABLE LOOKUPS
244
COMPUTE DIFFUSE LIGHTING COMPONENTS FOR Z
246
COMPUTE SPECULAR LIGHTING COMPONENTS FOR Z
248
COMPUTE POLE LIGHTING COMPONENTS FOR Z
250
COMPUTE EQUATOR LIGHTING COMPONENTS FOR Z
252
COMPUTE POLE CROSSING EQUATOR COMPONENTS FOR Z
FOR EACH PIXEL ON EACH SCAN LINE
END

FIG. 9

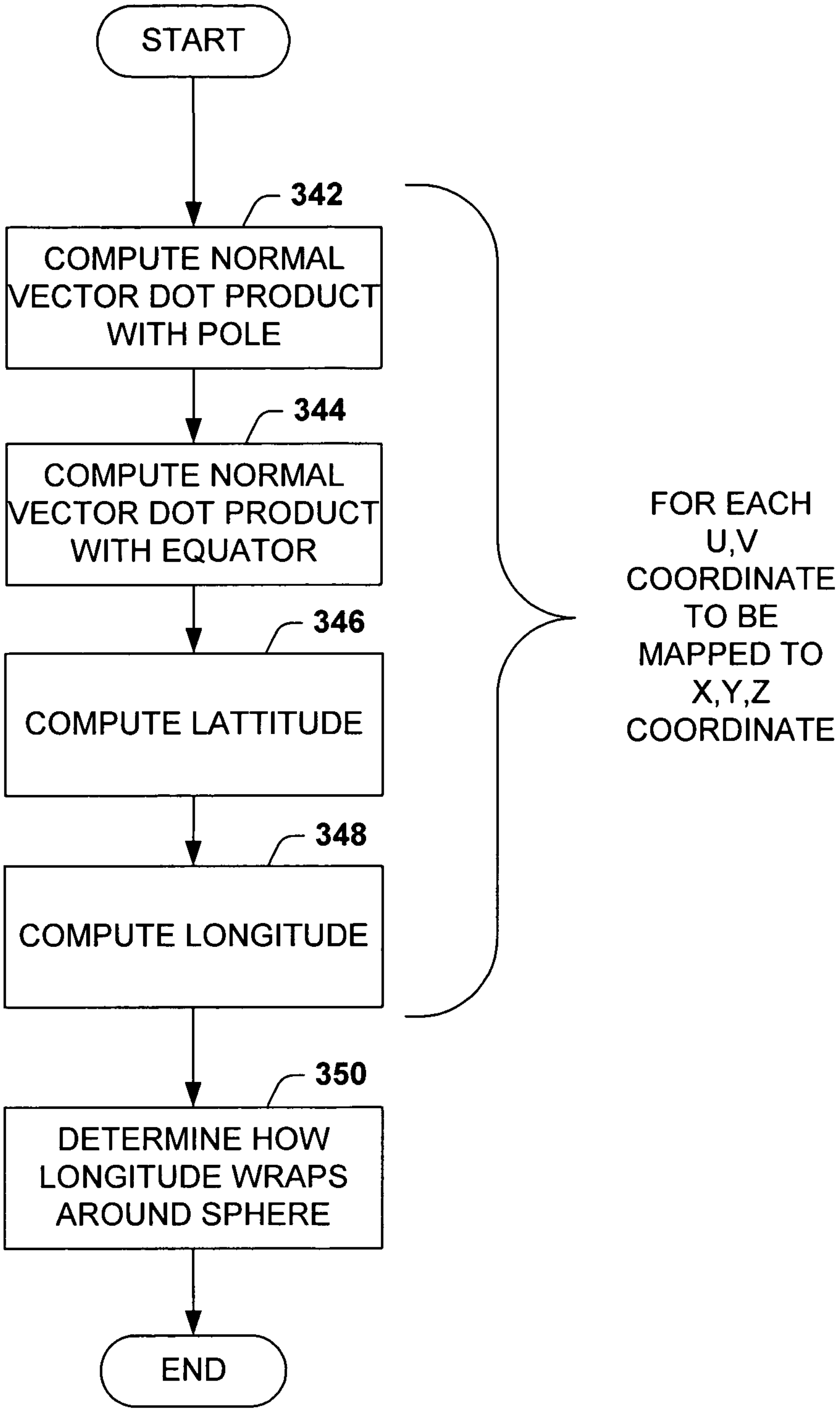
START
342
COMPUTE NORMAL VECTOR DOT PRODUCT WITH POLE
344
COMPUTE NORMAL VECTOR DOT PRODUCT WITH EQUATOR
346
COMPUTE LATTITUDE
348
COMPUTE LONGITUDE
FOR EACH U,V COORDINATE TO BE MAPPED TO X,Y,Z COORDINATE
350
DETERMINE HOW LONGITUDE WRAPS AROUND SPHERE
END

FIG. 10

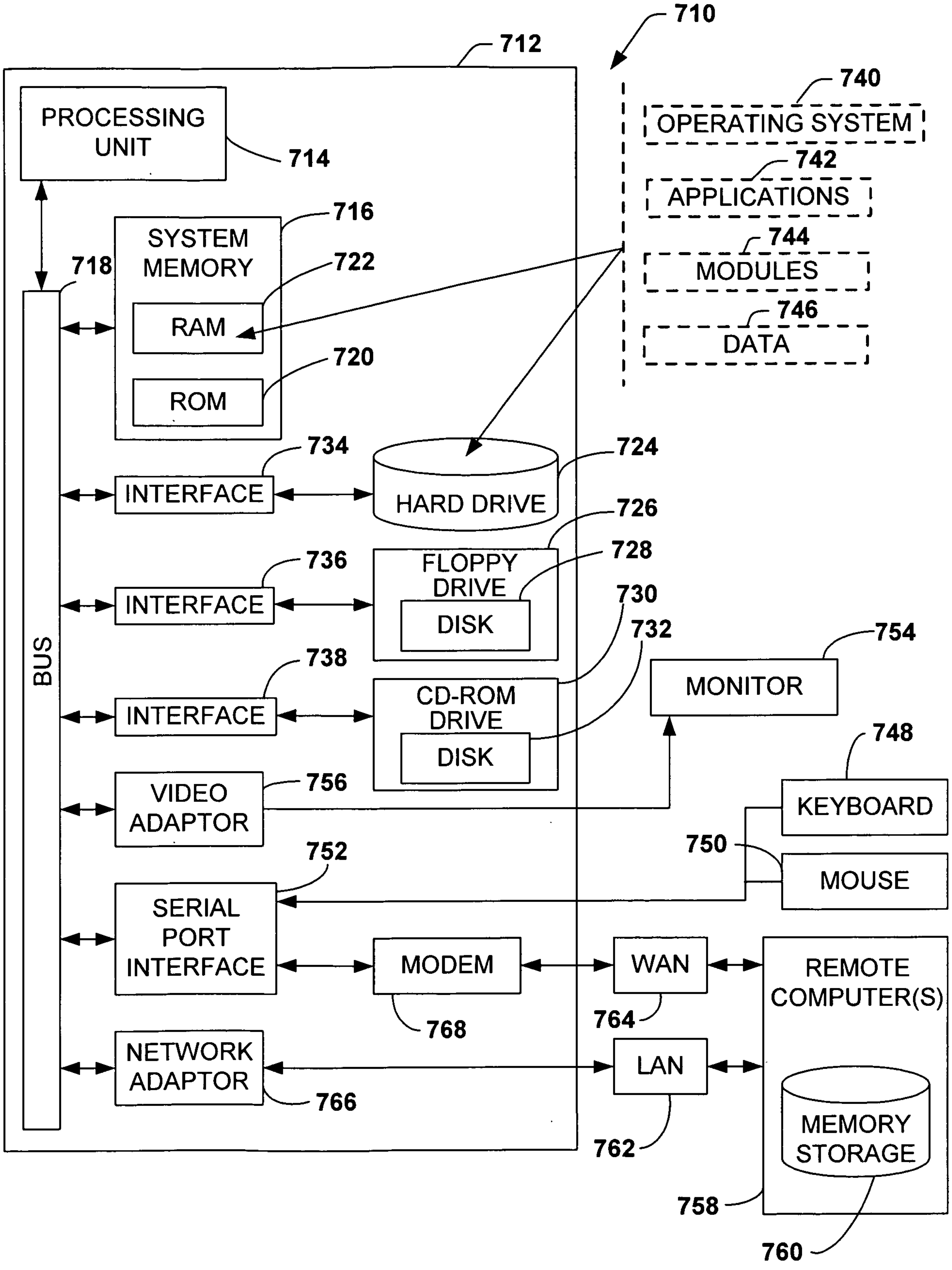
710
712
PROCESSING UNIT
714
SYSTEM MEMORY
716
718
722
RAM
720
ROM
BUS
740
OPERATING SYSTEM
742
APPLICATIONS
744
MODULES
746
DATA
734
INTERFACE
HARD DRIVE
724
726
FLOPPY DRIVE
728
DISK
736
INTERFACE
730
732
738
INTERFACE
CD-ROM DRIVE
DISK
754
MONITOR
VIDEO ADAPTOR
756
748
KEYBOARD
750
MOUSE
752
SERIAL PORT INTERFACE
MODEM
768
WAN
764
REMOTE COMPUTER(S)
NETWORK ADAPTOR
766
LAN
762
MEMORY STORAGE
758
760

FIG. 11